# Contents



# 1 Setting

## 1.1 vimrc

```
set nocp ai si nu et bs=2 mouse=a
set ts=2 sts=2 sw=2 hls showmatch
set ruler rulerformat=%17.(%l:%c%)
set noswapfile autoread wildmenu wildmode=list:longest
syntax on | colorscheme evening

map <F5> <ESC>:w<CR>:!g++ -g -Wall --std=c++0x -O2 %:r.cpp -o %:r && %:r < %:r.in > %:r.out<CR>
map <F6> <ESC>:w<CR>:!g++ -g -Wall --std=c++0x -O2 %:r.cpp -o %:r && %:r < %:r.in<CR>

map k gk
map j gj

map <C-h> <C-w>h
map <C-j> <C-w>j
map <C-k> <C-w>k
map <C-l> <C-w>l

map <C-t> :tabnew<CR>

```

```
command -nargs=1 PS :cd d:/ | :vi <args>.cpp | vs <args>.in | sp <args>.out
```

# 2 Math

## 2.1 Basic Arithmetic

```
typedef long long ll;
typedef unsigned long long ull;

// calculate lg2(a)
inline int lg2(ll a)
{
    return 63 - __builtin_clzll(a);
}

// calculate the number of 1-bits
inline int bitcount(ll a)
{
    return __builtin_popcountll(a);
}

// calculate ceil(a/b)
// |a|, |b| <= (2^63)-1 (does not dover -2^63)
ll ceildiv(ll a, ll b) {
    if (b < 0) return ceildiv(-a, -b);
    if (a < 0) return (-a) / b;
    return ((ull)a + (ull)b - 1ull) / b;
}

// calculate floor(a/b)
// |a|, |b| <= (2^63)-1 (does not cover -2^63)
ll floordiv(ll a, ll b) {
    if (b < 0) return floordiv(-a, -b);
    if (a >= 0) return a / b;
    return -(ll)(((ull)(-a) + b - 1) / b);
}

// calculate a*b % m
// x86-64 only
ll large_mod_mul(ll a, ll b, ll m)
{
    return ll((__int128)a*(__int128)b%m);
}

// calculate a*b % m
// |m| < 2^62, x86 available
// O(logb)
ll large_mod_mul(ll a, ll b, ll m)
{
    a %= m; b %= m; ll r = 0, v = a;
    while (b) {
        if (b&1) r = (r + v) % m;
        b >>= 1;
        v = (v << 1) % m;
    }
    return r;
}

// calculate n^k % m
ll modpow(ll n, ll k, ll m) {
    ll ret = 1;
    n %= m;
    while (k) {
        if (k & 1) ret = large_mod_mul(ret, n, m);
        n = large_mod_mul(n, n, m);
        k /= 2;
    }
    return ret;
}

// calculate gcd(a, b)
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

// find a pair (c, d) s.t. ac + bd = gcd(a, b)
pair<ll, ll> extended_gcd(ll a, ll b) {
    if (b == 0) return { 1, 0 };
    auto t = extended_gcd(b, a % b);
    return { t.second, t.first - t.second * (a / b) };
}

// find x in [0,m) s.t. ax === gcd(a, m) (mod m)
ll modinverse(ll a, ll m) {
    return (extended_gcd(a, m).first % m + m) % m;
}

// calculate modular inverse for 1 ~ n
void calc_range_modinv(int n, int mod, int ret[]) {
    ret[1] = 1;
    for (int i = 2; i <= n; ++i)
        ret[i] = (ll)(mod - mod/i) * ret[mod%i] % mod;
}
```

## 2.2 Sieve Methods : Prime, Divisor, Euler phi

```
// find prime numbers in 1 ~ n
// ret[x] = false  ->  x is prime
// O(n*loglogn)
void sieve(int n, bool ret[]) {
    for (int i = 2; i * i <= n; ++i)
        if (!ret[i])
            for (int j = i * i; j <= n; j += i)
                ret[i] = true;
}

// calculate number of divisors for 1 ~ n
```

```cpp
// when you need to calculate sum, change += 1 to += i
// O(n*logn)
void num_of_divisors(int n, int ret[]) {
    for (int i = 1; i <= n; ++i)
        for (int j = i; j <= n; j += i)
            ret[j] += 1;
}

// calculate euler totient function for 1 ~ n
// phi(n) = number of x s.t. 0 < x < n && gcd(n, x) = 1
// O(n*loglogn)
void euler_phi(int n, int ret[]) {
    for (int i = 1; i <= n; ++i) ret[i] = i;
    for (int i = 2; i <= n; ++i)
        if (ret[i] == i)
            for (int j = i; j <= n; j += i)
                ret[j] -= ret[j] / i;
}
```

## 2.3 Primality Test

```cpp
bool test_witness(ull a, ull n, ull s) {
    if (a >= n) a %= n;
    if (a <= 1) return true;
    ull d = n >> s;
    ull x = modpow(a, d, n);
    if (x == 1 || x == n-1) return true;
    while (s-- > 1) {
        x = large_mod_mul(x, x, n);
        if (x == 1) return false;
        if (x == n-1) return true;
    }
    return false;
}

// test whether n is prime
// based on miller-rabin test
// O(logn*logn)
bool is_prime(ull n) {
    if (n == 2) return true;
    if (n < 2 || n % 2 == 0) return false;

    ull d = n >> 1, s = 1;
    for(; (d&1) == 0; s++) d >>= 1;

#define T(a) test_witness(a##ull, n, s)
    if (n < 4759123141ull) return T(2) && T(7) && T(61);
    return T(2) && T(325) && T(9375) && T(28178)
        && T(450775) && T(9780504) && T(1795265022);
#undef T
}
```

## 2.4 Chinese Remainder Theorem

```cpp
// find x s.t.  x === a[0] (mod n[0])
//              === a[1] (mod n[1])
//              ...
// assumption: gcd(n[i], n[j]) = 1
ll chinese_remainder(ll* a, ll* n, int size) {
    if (size == 1) return *a;
    ll tmp = modinverse(n[0], n[1]);
    ll tmp2 = (tmp * (a[1] - a[0]) % n[1] + n[1]) % n[1];
    ll ora = a[1];
    ll tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2;
    n[1] *= n[0] / tgcd;
    ll ret = chinese_remainder(a + 1, n + 1, size - 1);
    n[1] /= n[0] / tgcd;
    a[1] = ora;
    return ret;
}
```

## 2.5 Modular Equation

$x \equiv a \pmod m$, $x \equiv b \pmod n$을 만족시키는 $x$를 구하는 방법.

$m$과 $n$을 소인수분해한 후 소수의 제곱꼴의 합동식들로 각각 쪼갠다. 이 때 특정 소수에 대하여 모순이 생기면 불가능한 경우고, 모든 소수에 대해서 모순이 생기지 않으면 전체 식을 CRT로 합치면 된다. 이제 $x \equiv x_1 \pmod{p^{k_1}}$과 $x \equiv x_2 \pmod{p^{k_2}}$가 모순이 생길 조건은 $k_1 \le k_2$라고 했을 때, $x_1 \not\equiv x_2 \pmod{p^{k_1}}$인 경우이다. 모순이 생기지 않았을 때 답을 구하려면 CRT로 합칠 때 $x \equiv x_2 \pmod{p^{k_2}}$만을 남기고 합쳐주면 된다.

## 2.6 Rational Number Class

```cpp
struct rational {
    long long p, q;

    void red() {
        if (q < 0) {
            p = -p;
            q = -q;
        }
        ll t = gcd((p >= 0 ? p : -p), q);
        p /= t;
        q /= t;
    }

    rational(): p(0), q(1) {}
    rational(long long p_): p(p_), q(1) {}
    rational(long long p_, long long q_): p(p_), q(q_) { red(); }

    bool operator==(const rational& rhs) const {
        return p == rhs.p && q == rhs.q;
    }
    bool operator!=(const rational& rhs) const {
```

```
        return p != rhs.p || q != rhs.q;
    }
    bool operator<(const rational& rhs) const {
        return p * rhs.q < rhs.p * q;
    }
    rational operator+(const rational& rhs) const {
        ll g = gcd(q, rhs.q);
        return rational(p * (rhs.q / g) + rhs.p * (q / g), (q / g) * rhs.q);
    }
    rational operator-(const rational& rhs) const {
        ll g = gcd(q, rhs.q);
        return rational(p * (rhs.q / g) - rhs.p * (q / g), (q / g) * rhs.q);
    }
    rational operator*(const rational& rhs) const {
        return rational(p * rhs.p, q * rhs.q);
    }
    rational operator/(const rational& rhs) const {
        return rational(p * rhs.q, q * rhs.p);
    }
};
```

## 2.7 Burnside's Lemma

경우의 수를 세는데, 특정 transform operation(회전, 반사, ..) 해서 같은 경우들은 하나로 친다. 전체 경우의 수는?

- 각 operation마다 이 operation을 했을 때 변하지 않는 경우의 수를 센다 (단, "아무것도 하지 않는다"라는 operation도 있어야 함!)

- 전체 경우의 수를 더한 후, operation의 수로 나눈다. (답이 맞다면 항상 나누어 떨어져야 한다)

## 2.8 Kirchoff's Theorem

그래프의 스패닝 트리의 개수를 구하는 정리.

무향 그래프의 Laplacian matrix $L$를 만든다. 이것은 (정점의 차수 대각 행렬) - (인접행렬) 이다. $L$에서 행과 열을 하나씩 제거한 것을 $L'$라 하자. 어느 행/열이든 관계 없다. 그래프의 스패닝 트리의 개수는 $det(L')$이다.

## 2.9 Fast Fourier Transform

```
void fft(int sign, int n, double *real, double *imag) {
    double theta = sign * 2 * pi / n;
    for (int m = n; m >= 2; m >>= 1, theta *= 2) {
        double wr = 1, wi = 0, c = cos(theta), s = sin(theta);
        for (int i = 0, mh = m >> 1; i < mh; ++i) {
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                double xr = real[j] - real[k], xi = imag[j] - imag[k];
                real[j] += real[k], imag[j] += imag[k];
                real[k] = wr * xr - wi * xi, imag[k] = wr * xi + wi * xr;
            }
            double _wr = wr * c - wi * s, _wi = wr * s + wi * c;
            wr = _wr, wi = _wi;
        }
    }
    for (int i = 1, j = 0; i < n; ++i) {
        for (int k = n >> 1; k > (j ^= k); k >>= 1);
        if (j < i) swap(real[i], real[j]), swap(imag[i], imag[j]);
    }
}
// Compute Poly(a)*Poly(b), write to r; Indexed from 0
// O(n*logn)
int mult(int *a, int n, int *b, int m, int *r) {
    const int maxn = 100;
    static double ra[maxn], rb[maxn], ia[maxn], ib[maxn];
    int fn = 1;
    while (fn < n + m) fn <<= 1; // n + m: interested length
    for (int i = 0; i < n; ++i) ra[i] = a[i], ia[i] = 0;
    for (int i = n; i < fn; ++i) ra[i] = ia[i] = 0;
    for (int i = 0; i < m; ++i) rb[i] = b[i], ib[i] = 0;
    for (int i = m; i < fn; ++i) rb[i] = ib[i] = 0;
    fft(1, fn, ra, ia);
    fft(1, fn, rb, ib);
    for (int i = 0; i < fn; ++i) {
        double real = ra[i] * rb[i] - ia[i] * ib[i];
        double imag = ra[i] * ib[i] + rb[i] * ia[i];
        ra[i] = real, ia[i] = imag;
    }
    fft(-1, fn, ra, ia);
    for (int i = 0; i < fn; ++i) r[i] = (int)floor(ra[i] / fn + 0.5);
    return fn;
}
```

## 2.10 Matrix Operations

```
const int MATSZ = 100;

inline bool is_zero(double a) { return fabs(a) < 1e-9; }

// out = A^(-1), returns det(A)
// A becomes invalid after call this
// O(n^3)
double inverse_and_det(int n, double A[][MATSZ], double out[][MATSZ]) {
    double det = 1;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) out[i][j] = 0;
        out[i][i] = 1;
    }
    for (int i = 0; i < n; i++) {
        if (is_zero(A[i][i])) {
            double maxv = 0;
```

```
            int maxid = -1;
            for (int j = i + 1; j < n; j++) {
                auto cur = fabs(A[j][i]);
                if (maxv < cur) {
                    maxv = cur;
                    maxid = j;
                }
            }
            if (maxid == -1 || is_zero(A[maxid][i])) return 0;
            for (int k = 0; k < n; k++) {
                A[i][k] += A[maxid][k];
                out[i][k] += out[maxid][k];
            }
        }
        det *= A[i][i];
        double coeff = 1.0 / A[i][i];
        for (int j = 0; j < n; j++) A[i][j] *= coeff;
        for (int j = 0; j < n; j++) out[i][j] *= coeff;
        for (int j = 0; j < n; j++) if (j != i) {
            double mp = A[j][i];
            for (int k = 0; k < n; k++) A[j][k] -= A[i][k] * mp;
            for (int k = 0; k < n; k++) out[j][k] -= out[i][k] * mp;
        }
    }
    return det;
}
```

## 2.11 Gaussian Elimination

```
const double EPS = 1e-10;
typedef vector<vector<double>> VVD;

// Gauss-Jordan elimination with full pivoting.
// solving systems of linear equations (AX=B)
// INPUT:    a[][] = an n*n matrix
//           b[][] = an n*m matrix
// OUTPUT:   X      = an n*m matrix (stored in b[][])
//           A^{-1} = an n*n matrix (stored in a[][])
// O(n^3)
bool gauss_jordan(VVD& a, VVD& b) {
    const int n = a.size();
    const int m = b[0].size();
    vector<int> irow(n), icol(n), ipiv(n);

    for (int i = 0; i < n; i++) {
        int pj = -1, pk = -1;
        for (int j = 0; j < n; j++) if (!ipiv[j])
            for (int k = 0; k < n; k++) if (!ipiv[k])
                if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk =
                  k; }
        if (fabs(a[pj][pk]) < EPS) return false; // matrix is singular
        ipiv[pk]++;
        swap(a[pj], a[pk]);
        swap(b[pj], b[pk]);
        irow[i] = pj;
        icol[i] = pk;

        double c = 1.0 / a[pk][pk];
        a[pk][pk] = 1.0;
        for (int p = 0; p < n; p++) a[pk][p] *= c;
        for (int p = 0; p < m; p++) b[pk][p] *= c;
        for (int p = 0; p < n; p++) if (p != pk) {
            c = a[p][pk];
            a[p][pk] = 0;
            for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
            for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
        }
    }
    for (int p = n - 1; p >= 0; p--) if (irow[p] != icol[p]) {
        for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
    }
    return true;
}
```

## 2.12 Simplex Algorithm

```
// Two-phase simplex algorithm for solving linear programs of the form
//     maximize     c^T x
//     subject to   Ax <= b
//                   x >= 0
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).
typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
const double EPS = 1e-9;

struct LPSolver {
    int m, n;
    VI B, N;
    VVD D;

    LPSolver(const VVD& A, const VD& b, const VD& c) :
        m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
        for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j
          ];
        for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n + 1] =
          b[i]; }
        for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
        N[n] = -1; D[m + 1][n] = 1;
    }
```

```
    void pivot(int r, int s) {
        double inv = 1.0 / D[r][s];
        for (int i = 0; i < m + 2; i++) if (i != r)
            for (int j = 0; j < n + 2; j++) if (j != s)
                D[i][j] -= D[r][j] * D[i][s] * inv;
        for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
        for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
        D[r][s] = inv;
        swap(B[r], N[s]);
    }

    bool simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
        while (true) {
            int s = -1;
            for (int j = 0; j <= n; j++) {
                if (phase == 2 && N[j] == -1) continue;
                if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] <
                   N[s]) s = j;
            }
            if (D[x][s] > -EPS) return true;
            int r = -1;
            for (int i = 0; i < m; i++) {
                if (D[i][s] < EPS) continue;
                if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
                    (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] <
                       B[r]) r = i;
            }
            if (r == -1) return false;
            pivot(r, s);
        }
    }

    double solve(VD& x) {
        int r = 0;
        for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
        if (D[r][n + 1] < -EPS) {
            pivot(r, n);
            if (!simplex(1) || D[m + 1][n + 1] < -EPS)
                return -numeric_limits<double>::infinity();
            for (int i = 0; i < m; i++) if (B[i] == -1) {
                int s = -1;
                for (int j = 0; j <= n; j++)
                    if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[
                      j] < N[s]) s = j;
                pivot(i, s);
            }
        }
        if (!simplex(2))
            return numeric_limits<double>::infinity();
        x = VD(n);
        for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
        return D[m][n + 1];
    }
};
```

# 3 Data Structure

## 3.1 Order statistic tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <functional>
#include <iostream>
using namespace __gnu_pbds;
using namespace std;

// tree<key_type, value_type(set if null), comparator, ...>
using ordered_set = tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;

int main()
{
    ordered_set X;
    for (int i = 1; i < 10; i += 2) X.insert(i); // 1 3 5 7 9
    cout << boolalpha;
    cout << *X.find_by_order(2) << endl; // 5
    cout << *X.find_by_order(4) << endl; // 9
    cout << (X.end() == X.find_by_order(5)) << endl; // true

    cout << X.order_of_key(-1) << endl; // 0
    cout << X.order_of_key(1) << endl; // 0
    cout << X.order_of_key(4) << endl; // 2
    X.erase(3);
    cout << X.order_of_key(4) << endl; // 1
    for (int t : X) printf("%d␣", t); // 1 5 7 9
}
```

## 3.2 Fenwick Tree

```
const int TSIZE = 100000;
int tree[TSIZE + 1];

// Returns the sum from index 1 to p, inclusive
int query(int p) {
    int ret = 0;
    for (; p > 0; p -= p & -p) ret += tree[p];
    return ret;
}

// Adds val to element with index pos
void add(int p, int val) {
    for (; p <= TSIZE; p += p & -p) tree[p] += val;
}
```

## 3.3 Segment Tree with Lazy Propagation

```cpp
// example implementation of sum tree
const int TSIZE = 131072; // always 2^k form && n <= TSIZE
int segtree[TSIZE * 2], prop[TSIZE * 2];
void seg_init(int nod, int l, int r) {
    if (l == r) segtree[nod] = dat[l];
    else {
        int m = (l + r) >> 1;
        seg_init(nod << 1, l, m);
        seg_init(nod << 1 | 1, m + 1, r);
        segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];
    }
}
void seg_relax(int nod, int l, int r) {
    if (prop[nod] == 0) return;
    if (l < r) {
        int m = (l + r) >> 1;
        segtree[nod << 1] += (m - l + 1) * prop[nod];
        prop[nod << 1] += prop[nod];
        segtree[nod << 1 | 1] += (r - m) * prop[nod];
        prop[nod << 1 | 1] += prop[nod];
    }
    prop[nod] = 0;
}
int seg_query(int nod, int l, int r, int s, int e) {
    if (r < s || e < l) return 0;
    if (s <= l && r <= e) return segtree[nod];
    seg_relax(nod, l, r);
    int m = (l + r) >> 1;
    return seg_query(nod << 1, l, m, s, e) + seg_query(nod << 1 | 1, m + 1, r, s
      , e);
}
void seg_update(int nod, int l, int r, int s, int e, int val) {
    if (r < s || e < l) return;
    if (s <= l && r <= e) {
        segtree[nod] += (r - l + 1) * val;
        prop[nod] += val;
        return;
    }
    seg_relax(nod, l, r);
    int m = (l + r) >> 1;
    seg_update(nod << 1, l, m, s, e, val);
    seg_update(nod << 1 | 1, m + 1, r, s, e, val);
    segtree[nod] = segtree[nod << 1] + segtree[nod << 1 | 1];
}
// usage:
// seg_update(1, 0, n - 1, qs, qe, val);
// seg_query(1, 0, n - 1, qs, qe);
```

## 3.4 Persistent Segment Tree

```cpp
// persistent segment tree impl: sum tree
namespace pstree {
    typedef int val_t;
    const int DEPTH = 18;
    const int TSIZE = 1 << 18;
    const int MAX_QUERY = 262144;

    struct node {
        val_t v;
        node *l, *r;
    } npoll[TSIZE * 2 + MAX_QUERY * (DEPTH + 1)];

    int pptr, last_q;

    node *head[MAX_QUERY + 1];
    int q[MAX_QUERY + 1];
    int lqidx;

    void init() {
        // zero-initialize, can be changed freely
        memset(&npoll[TSIZE - 1], 0, sizeof(node) * TSIZE);

        for (int i = TSIZE - 2; i >= 0; i--) {
            npoll[i].v = 0;
            npoll[i].l = &npoll[i*2+1];
            npoll[i].r = &npoll[i*2+2];
        }

        head[0] = &npoll[0];
        last_q = 0;
        pptr = 2 * TSIZE - 1;
        q[0] = 0;
        lqidx = 0;
    }

    // update val to pos at time t
    // 0 <= t <= MAX_QUERY, 0 <= pos < TSIZE
    void update(int pos, int val, int t, int prev) {
        head[++last_q] = &npoll[pptr++];
        node *old = head[q[prev]], *now = head[last_q];
        while (lqidx < t) q[lqidx++] = q[prev];
        q[t] = last_q;

        int flag = 1 << DEPTH;
        for (;;) {
            now->v = old->v + val;
            flag >>= 1;
            if (flag==0) {
                now->l = now->r = nullptr; break;
            }
            if (flag & pos) {
                now->l = old->l;
                now->r = &npoll[pptr++];
                now = now->r, old = old->r;
            } else {
                now->r = old->r;
                now->l = &npoll[pptr++];
                now = now->l, old = old->l;
            }
```

```
        }
    }

    val_t query(int s, int e, int l, int r, node *n) {
        if (s == l && e == r) return n->v;
        int m = (l + r) / 2;
        if (m >= e) return query(s, e, l, m, n->l);
        else if (m < s) return query(s, e, m + 1, r, n->r);
        else return query(s, m, l, m, n->l) + query(m + 1, e, m + 1, r, n->r);
    }

    // query summation of [s, e] at time t
    val_t query(int s, int e, int t) {
        s = max(0, s); e = min(TSIZE - 1, e);
        if (s > e) return 0;
        return query(s, e, 0, TSIZE - 1, head[q[t]]);
    }
}
```

## 3.5 Splay Tree

```
// example : https://www.acmicpc.net/problem/13159
struct node {
    node* l, * r, * p;
    int cnt, min, max, val;
    long long sum;
    bool inv;
    node(int _val) :
        cnt(1), sum(_val), min(_val), max(_val), val(_val), inv(false),
        l(nullptr), r(nullptr), p(nullptr) {
    }
};
node* root;

void update(node* x) {
    x->cnt = 1;
    x->sum = x->min = x->max = x->val;
    if (x->l) {
        x->cnt += x->l->cnt;
        x->sum += x->l->sum;
        x->min = min(x->min, x->l->min);
        x->max = max(x->max, x->l->max);
    }
    if (x->r) {
        x->cnt += x->r->cnt;
        x->sum += x->r->sum;
        x->min = min(x->min, x->r->min);
        x->max = max(x->max, x->r->max);
    }
}

void rotate(node* x) {
    node* p = x->p;
    node* b = nullptr;
    if (x == p->l) {
        p->l = b = x->r;
        x->r = p;
    }
    else {
        p->r = b = x->l;
        x->l = p;
    }
    x->p = p->p;
    p->p = x;
    if (b) b->p = p;
    x->p ? (p == x->p->l ? x->p->l : x->p->r) = x : (root = x);
    update(p);
    update(x);
}

// make x into root
void splay(node* x) {
    while (x->p) {
        node* p = x->p;
        node* g = p->p;
        if (g) rotate((x == p->l) == (p == g->l) ? p : x);
        rotate(x);
    }
}

void relax_lazy(node* x) {
    if (!x->inv) return;
    swap(x->l, x->r);
    x->inv = false;
    if (x->l) x->l->inv = !x->l->inv;
    if (x->r) x->r->inv = !x->r->inv;
}

// find kth node in splay tree
void find_kth(int k) {
    node* x = root;
    relax_lazy(x);
    while (true) {
        while (x->l && x->l->cnt > k) {
            x = x->l;
            relax_lazy(x);
        }
        if (x->l) k -= x->l->cnt;
        if (!k--) break;
        x = x->r;
        relax_lazy(x);
    }
    splay(x);
}

// collect [l, r] nodes into one subtree and return its root
node* interval(int l, int r) {
    find_kth(l - 1);
    node* x = root;
```

```
    root = x->r;
    root->p = nullptr;
    find_kth(r - l + 1);
    x->r = root;
    root->p = x;
    root = x;
    return root->r->l;
}

void traverse(node* x) {
    relax_lazy(x);
    if (x->l) {
        traverse(x->l);
    }
    // do something
    if (x->r) {
        traverse(x->r);
    }
}

void uptree(node* x) {
    if (x->p) {
        uptree(x->p);
    }
    relax_lazy(x);
}
```

### 3.6 Link/Cut Tree

# 4 DP

## 4.1 Convex Hull Optimization

### 4.1.1 requirement

$O(n^2) \rightarrow O(n \log n)$

조건 1) DP 점화식 꼴

$D[i] = \min_{j<i}(D[j] + b[j] * a[i])$

조건 2) $b[j] \leq b[j+1]$

특수조건) $a[i] \leq a[i+1]$ 도 만족하는 경우, 마지막 쿼리의 위치를 저장해두면 이분검색이 필요없어지기 때문에 amortized $O(n)$ 에 해결할 수 있음

### 4.1.2 Source Code

```
//O(n^3) -> O(n^2)

#define sz 100001
long long s[sz];
long long dp[2][sz];
//deque {index, x pos }
int dqi[sz];
long long dqm[sz];
//pointer to deque
int ql,qr;
//dp[i][j] = max(dp[i][k] + s[j]*s[k] - s[k]^2)
//let y = dp[i][j], x = s[j] -> y = max(s[k]*x + dp[i][k] - s[k]^2);

//push new value to deque
//i = index, x = current x pos
void setq(int i, int x)
{
    //a1,b1 = prv line, a2,b2 = new line
    int a1, a2 = s[i];
    long long b1, b2 = dp[0][i] - s[i] * s[i], r;
    //renew deque
    while (qr>=ql)
    {
        //last line enqueued
        a1 = s[dqi[qr]];
        b1 = dp[0][dqi[qr]] - s[dqi[qr]] * s[dqi[qr]];
        //tie breaking to newer one
        if (a1 == a2)
        {
            dqi[qr] = i;
            return;
        }
        // x intersection between last line and new line
        r = (b1 - b2) / (a2 - a1);
        if ((b1 - b2) % (a2 - a1)) r++;
        //last line is not needed
        if (r <= dqm[qr])
        {
            qr--;
        }
        else break;
    }
    if (r < 0) r = 0;
    //push back new line
    if (dqm[qr] < s[n - 1] && r <= s[n - 1])
    {
        dqi[++qr] = i;
        dqm[qr] = r;
    }
    //discard old lines
    while (qr-ql && dqm[ql+1] <= x)
    {
        ql++;
    }
}

int main()
```

```
{
    for (int j = 0; j < k; j++)
    {
        ql = 0;
        qr = 1;
        dqi[0] = dqm[0] = 0;
        for (int i = 1; i < n; i++)
        {
            //get line used by current x pos
            setq(i, s[i]);
            //Line index to use
            int g = dqi[ql];
            //set dp value
            dp[1][i] = dp[0][g] + s[g] * (s[i] - s[g]);
        }
        for (int i = 0; i < n; i++)
        {
            dp[0][i] = dp[1][i];
            dp[1][i] = 0;
        }
    }
}
```

### 4.2 Divide & Conquer Optimization

$O(kn^2) \rightarrow O(kn \log n)$

조건 1) DP 점화식 꼴

$D[t][i] = \min_{j<i}(D[t-1][j] + C[j][i])$

조건 2) $A[t][i]$는 $D[t][i]$의 답이 되는 최소의 $j$라 할 때, 아래의 부등식을 만족해야 함

$A[t][i] \le A[t][i+1]$

조건 2-1) 비용$C$가 다음의 사각부등식을 만족하는 경우도 조건 2)를 만족하게 됨

$C[a][c] + C[b][d] \le C[a][d] + C[b][c] \ \ (a \le b \le c \le d)$

### 4.3 Knuth Optimization

$O(n^3) \rightarrow O(n^2)$

조건 1) DP 점화식 꼴

$D[i][j] = \min_{i<k<j}(D[i][k] + D[k][j]) + C[i][j]$

조건 2) 사각 부등식

$C[a][c] + C[b][d] \le C[a][d] + C[b][c] \ \ (a \le b \le c \le d)$

조건 3) 단조성

$C[b][c] \le C[a][d] \ \ (a \le b \le c \le d)$

결론) 조건 2, 3을 만족한다면 $A[i][j]$를 $D[i][j]$의 답이 되는 최소의 $k$라 할 때, 아래의 부등식을 만족하게 됨

$A[i][j-1] \le A[i][j] \le A[i+1][j]$

3중 루프를 돌릴 때 위 조건을 이용하면 최종적으로 시간복잡도가 $O(n^2)$ 이 됨

## 5 Graph

### 5.1 SCC (Tarjan)

```
const int MAXN = 100;
vector<int> graph[MAXN];
int up[MAXN], visit[MAXN], vtime;
vector<int> stk;
int scc_idx[MAXN], scc_cnt;

void dfs(int nod) {
    up[nod] = visit[nod] = ++vtime;
    stk.push_back(nod);
    for (int next : graph[nod]) {
        if (visit[next] == 0) {
            dfs(next);
            up[nod] = min(up[nod], up[next]);
        }
        else if (scc_idx[next] == 0)
            up[nod] = min(up[nod], visit[next]);
    }
    if (up[nod] == visit[nod]) {
        ++scc_cnt;
        int t;
        do {
            t = stk.back();
            stk.pop_back();
            scc_idx[t] = scc_cnt;
        } while (!stk.empty() && t != nod);
    }
}

// find SCCs in given directed graph
// O(V+E)
void get_scc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    scc_cnt = 0;
    memset(scc_idx, 0, sizeof(scc_idx));
    for (int i = 0; i < n; ++i)
        if (visit[i] == 0) dfs(i);
}
```

## 5.2 SCC (Kosaraju)

```cpp
const int MAXN = 100;
vector<int> graph[MAXN], grev[MAXN];
int visit[MAXN], vcnt;
int scc_idx[MAXN], scc_cnt;
vector<int> emit;

void dfs(int nod, vector<int> graph[]) {
    visit[nod] = vcnt;
    for (int next : graph[nod]) {
        if (visit[next] == vcnt) continue;
        dfs(next, graph);
    }
    emit.push_back(nod);
}

// find SCCs in given graph
// O(V+E)
void get_scc() {
    scc_cnt = 0;
    vcnt = 1;
    emit.clear();
    memset(visit, 0, sizeof(visit));

    for (int i = 0; i < n; i++) {
        if (visit[i] == vcnt) continue;
        dfs(i, graph);
    }

    ++vcnt;
    for (auto st : vector<int>(emit.rbegin(), emit.rend())) {
        if (visit[st] == vcnt) continue;
        emit.clear();
        dfs(st, grev);
        ++scc_cnt;
        for (auto node : emit)
            scc_idx[node] = scc_cnt;
    }
}
```

## 5.3 2-SAT

$(b_x \vee b_y) \wedge (\neg b_x \vee b_z) \wedge (b_z \vee \neg b_x) \wedge \cdots$ 같은 form을 2-CNF라고 함. 주어진 2-CNF 식을 참으로 하는 $\{b_1, b_2, \cdots\}$ 가 존재하는지, 존재한다면 그 값은 무엇인지 구하는 문제를 2-SAT이라 함.

boolean variable $b_i$ 마다 $b_i$를 나타내는 정점, $\neg b_i$를 나타내는 정점 2개를 만듦. 각 clause $b_i \vee b_j$ 마다 $\neg b_i \to b_j$, $\neg b_j \to b_i$ 이렇게 edge를 이어줌. 그렇게 만든 그래프에서 SCC를 다 구함. 어떤 SCC 안에 $b_i$ 와 $\neg b_i$가 같이 포함되어있다면 해가 존재하지 않음. 아니라면 해가 존재함.

해가 존재할 때 구체적인 해를 구하는 방법. 위에서 SCC를 구하면서 SCC DAG를 만들어 준다. 거기서 위상정렬을 한 후, 앞에서부터 SCC를 하나씩 봐준다. 현재 보고있는 SCC에 $b_i$가 속해있는데 얘가 $\neg b_i$보다 먼저 등장했다면 $b_i = \text{false}$, 반대의 경우라면 $b_i = \text{true}$, 이미 값이 assign되었다면 pass.

## 5.4 BCC, Cut vertex, Bridge

```cpp
const int MAXN = 100;
vector<pair<int, int>> graph[MAXN];  // { next vertex id, edge id }
int up[MAXN], visit[MAXN], vtime;
vector<pair<int, int>> stk;

int is_cut[MAXN];           // v is cut vertex if  is_cut[v] > 0
vector<int> bridge;         // list of edge ids
vector<int> bcc_idx[MAXN];  // list of bccids for vertex i
int bcc_cnt;

void dfs(int nod, int par_edge) {
    up[nod] = visit[nod] = ++vtime;
    int child = 0;
    for (const auto& e : graph[nod]) {
        int next = e.first, edge_id = e.second;
        if (edge_id == par_edge) continue;
        if (visit[next] == 0) {
            stk.push_back({ nod, next });
            ++child;
            dfs(next, edge_id);
            if (up[next] == visit[next]) bridge.push_back(edge_id);
            if (up[next] >= visit[nod]) {
                ++bcc_cnt;
                do {
                    auto last = stk.back();
                    stk.pop_back();
                    bcc_idx[last.second].push_back(bcc_cnt);
                    if (last == pair<int, int>{ nod, next }) break;
                } while (!stk.empty());
                bcc_idx[nod].push_back(bcc_cnt);
                is_cut[nod]++;
            }
            up[nod] = min(up[nod], up[next]);
        }
        else
            up[nod] = min(up[nod], visit[next]);
    }
    if (par_edge == -1 && is_cut[nod] == 1)
        is_cut[nod] = 0;
}

// find BCCs & cut vertexs & bridges in undirected graph
// O(V+E)
void get_bcc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    memset(is_cut, 0, sizeof(is_cut));
```

```
    bridge.clear();
    for (int i = 0; i < n; ++i) bcc_idx[i].clear();
    bcc_cnt = 0;
    for (int i = 0; i < n; ++i) {
        if (visit[i] == 0)
            dfs(i, -1);
    }
}
```

## 5.5 Shortest Path Faster Algorithm

```
// shortest path faster algorithm
// average for random graph : O(E) , worst : O(VE)

const int MAXN = 20001;
const int INF = 100000000;
int n, m;
vector<pair<int, int>> graph[MAXN];
bool inqueue[MAXN];
int dist[MAXN];

void spfa(int st) {
    for (int i = 0; i < n; ++i) {
        dist[i] = INF;
    }
    dist[st] = 0;

    queue<int> q;
    q.push(st);
    inqueue[st] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inqueue[u] = false;
        for (auto& e : graph[u]) {
            if (dist[u] + e.second < dist[e.first]) {
                dist[e.first] = dist[u] + e.second;
                if (!inqueue[e.first]) {
                    q.push(e.first);
                    inqueue[e.first] = true;
                }
            }
        }
    }
}
```

## 5.6 Lowest Common Ancestor

```
const int MAXN = 100;
const int MAXLN = 9;
vector<int> tree[MAXN];
int depth[MAXN];
int par[MAXLN][MAXN];

void dfs(int nod, int parent) {
    for (int next : tree[nod]) {
        if (next == parent) continue;
        depth[next] = depth[nod] + 1;
        par[0][next] = nod;
        dfs(next, nod);
    }
}

void prepare_lca() {
    const int root = 0;
    dfs(root, -1);
    par[0][root] = root;
    for (int i = 1; i < MAXLN; ++i)
        for (int j = 0; j < n; ++j)
            par[i][j] = par[i - 1][par[i - 1][j]];
}

// find lowest common ancestor in tree between u & v
// assumption : must call 'prepare_lca' once before call this
// O(logV)
int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    if (depth[u] > depth[v]) {
        for (int i = MAXLN - 1; i >= 0; --i)
            if (depth[u] - (1 << i) >= depth[v])
                u = par[i][u];
    }
    if (u == v) return u;
    for (int i = MAXLN - 1; i >= 0; --i) {
        if (par[i][u] != par[i][v]) {
            u = par[i][u];
            v = par[i][v];
        }
    }
    return par[0][u];
}
```

## 5.7 Heavy-Light Decomposition

```
// heavy-light decomposition
//
// hld h;
// insert edges to tree[0~n-1];
// h.init(n);
// h.decompose(root);
// h.hldquery(u, v); // edges from u to v
struct hld {
    static const int MAXLN = 18;
    static const int MAXN = 1 << (MAXLN - 1);
    vector<int> tree[MAXN];
    int subsize[MAXN], depth[MAXN], pa[MAXLN][MAXN];

```

```
    int chead[MAXN], cidx[MAXN];
    int lchain;
    int flatpos[MAXN + 1], fptr;

    void dfs(int u, int par) {
        pa[0][u] = par;
        subsize[u] = 1;
        for (int v : tree[u]) {
            if (v == pa[0][u]) continue;
            depth[v] = depth[u] + 1;
            dfs(v, u);
            subsize[u] += subsize[v];
        }
    }

    void init(int size)
    {
        lchain = fptr = 0;
        dfs(0, -1);
        memset(chead, -1, sizeof(chead));

        for (int i = 1; i < MAXLN; i++) {
            for (int j = 0; j < size; j++) {
                if (pa[i - 1][j] != -1) {
                    pa[i][j] = pa[i - 1][pa[i - 1][j]];
                }
            }
        }
    }

    void decompose(int u) {
        if (chead[lchain] == -1) chead[lchain] = u;
        cidx[u] = lchain;
        flatpos[u] = ++fptr;

        int maxchd = -1;
        for (int v : tree[u]) {
            if (v == pa[0][u]) continue;
            if (maxchd == -1 || subsize[maxchd] < subsize[v]) maxchd = v;
        }
        if (maxchd != -1) decompose(maxchd);

        for (int v : tree[u]) {
            if (v == pa[0][u] || v == maxchd) continue;
            ++lchain; decompose(v);
        }
    }

    int lca(int u, int v) {
        if (depth[u] < depth[v]) swap(u, v);

        int logu;
        for (logu = 1; 1 << logu <= depth[u]; logu++);
        logu--;

        int diff = depth[u] - depth[v];
        for (int i = logu; i >= 0; --i) {
            if ((diff >> i) & 1) u = pa[i][u];
        }
        if (u == v) return u;

        for (int i = logu; i >= 0; --i) {
            if (pa[i][u] != pa[i][v]) {
                u = pa[i][u];
                v = pa[i][v];
            }
        }
        return pa[0][u];
    }

    // TODO: implement query functions
    inline int query(int s, int e) {
        return 0;
    }

    int subquery(int u, int v, int t) {
        int uchain, vchain = cidx[v];
        int ret = 0;
        for (;;) {
            uchain = cidx[u];
            if (uchain == vchain) {
                ret += query(flatpos[v], flatpos[u]);
                break;
            }

            ret += query(flatpos[chead[uchain]], flatpos[u]);
            u = pa[0][chead[uchain]];
        }
        return ret;
    }

    inline int hldquery(int u, int v) {
        int p = lca(u, v);
        return subquery(u, p) + subquery(v, p) - query(flatpos[p], flatpos[p]);
    }
};
```

## 5.8 Bipartite Matching (Hopcroft-Karp)

```
// in: n, m, graph
// out: match, matched
// vertex cover: (reached[0][left_node] == 0) || (reached[1][right_node] == 1)
// O(E*sqrt(V))
struct BipartiteMatching {
    int n, m;
    vector<vector<int>> graph;
    vector<int> matched, match, edgeview, level;
    vector<int> reached[2];
    BipartiteMatching(int n, int m) : n(n), m(m), graph(n), matched(m, -1),
```

```
    match(n, -1) {}

bool assignLevel() {
    bool reachable = false;
    level.assign(n, -1);
    reached[0].assign(n, 0);
    reached[1].assign(m, 0);
    queue<int> q;
    for (int i = 0; i < n; i++) {
        if (match[i] == -1) {
            level[i] = 0;
            reached[0][i] = 1;
            q.push(i);
        }
    }
    while (!q.empty()) {
        auto cur = q.front(); q.pop();
        for (auto adj : graph[cur]) {
            reached[1][adj] = 1;
            auto next = matched[adj];
            if (next == -1) {
                reachable = true;
            }
            else if (level[next] == -1) {
                level[next] = level[cur] + 1;
                reached[0][next] = 1;
                q.push(next);
            }
        }
    }
    return reachable;
}

int findpath(int nod) {
    for (int &i = edgeview[nod]; i < graph[nod].size(); i++) {
        int adj = graph[nod][i];
        int next = matched[adj];
        if (next >= 0 && level[next] != level[nod] + 1) continue;
        if (next == -1 || findpath(next)) {
            match[nod] = adj;
            matched[adj] = nod;
            return 1;
        }
    }
    return 0;
}

int solve() {
    int ans = 0;
    while (assignLevel()) {
        edgeview.assign(n, 0);
        for (int i = 0; i < n; i++)
            if (match[i] == -1)
                ans += findpath(i);
    }
        return ans;
    }
};
```

## 5.9 Maximum Flow (Dinic)

```
// usage:
// MaxFlowDinic::init(n);
// MaxFlowDinic::add_edge(0, 1, 100, 100); // for bidirectional edge
// MaxFlowDinic::add_edge(1, 2, 100); // directional edge
// result = MaxFlowDinic::solve(0, 2); // source -> sink
// graph[i][edgeIndex].res -> residual
//
// in order to find out the minimum cut, use `l'.
// if l[i] == 0, i is unrechable.
//
// O(V*V*E)
// with unit capacities, O(min(V^(2/3), E^(1/2)) * E)
struct MaxFlowDinic {
    typedef int flow_t;
    struct Edge {
        int next;
        int inv; /* inverse edge index */
        flow_t res; /* residual */
    };
    int n;
    vector<vector<Edge>> graph;
    vector<int> q, l, start;

    void init(int _n) {
        n = _n;
        graph.resize(n);
        for (int i = 0; i < n; i++) graph[i].clear();
    }
    void add_edge(int s, int e, flow_t cap, flow_t caprev = 0) {
        Edge forward{ e, graph[e].size(), cap };
        Edge reverse{ s, graph[s].size(), caprev };
        graph[s].push_back(forward);
        graph[e].push_back(reverse);
    }
    bool assign_level(int source, int sink) {
        int t = 0;
        memset(&l[0], 0, sizeof(l[0]) * l.size());
        l[source] = 1;
        q[t++] = source;
        for (int h = 0; h < t && !l[sink]; h++) {
            int cur = q[h];
            for (const auto& e : graph[cur]) {
                if (l[e.next] || e.res == 0) continue;
                l[e.next] = l[cur] + 1;
                q[t++] = e.next;
            }
        }
        return l[sink] != 0;
```

```
    }
    flow_t block_flow(int cur, int sink, flow_t current) {
        if (cur == sink) return current;
        for (int& i = start[cur]; i < graph[cur].size(); i++) {
            auto& e = graph[cur][i];
            if (e.res == 0 || l[e.next] != l[cur] + 1) continue;
            if (flow_t res = block_flow(e.next, sink, min(e.res, current))) {
                e.res -= res;
                graph[e.next][e.inv].res += res;
                return res;
            }
        }
        return 0;
    }
    flow_t solve(int source, int sink) {
        q.resize(n);
        l.resize(n);
        start.resize(n);
        flow_t ans = 0;
        while (assign_level(source, sink)) {
            memset(&start[0], 0, sizeof(start[0]) * n);
            while (flow_t flow = block_flow(source, sink, numeric_limits<flow_t
              >::max()))
                ans += flow;
        }
        return ans;
    }
};
```

## 5.10 Min-cost Maximum Flow

```
// precondition: there is no negative cycle.
// usage:
//  MinCostFlow mcf(n);
//  for(each edges) mcf.addEdge(from, to, cost, capacity);
//  mcf.solve(source, sink); // min cost max flow
//  mcf.solve(source, sink, 0); // min cost flow
//  mcf.solve(source, sink, goal_flow); // min cost flow with total_flow >=
  goal_flow if possible
struct MinCostFlow
{
    typedef int cap_t;
    typedef int cost_t;

    bool iszerocap(cap_t cap) { return cap == 0; }

    struct edge {
        int target;
        cost_t cost;
        cap_t residual_capacity;
        cap_t orig_capacity;
        size_t revid;
    };

    int n;
    vector<vector<edge>> graph;
    vector<cost_t> pi;
    bool needNormalize, ranbefore;
    int lastStart;

    MinCostFlow(int n) : graph(n), n(n), pi(n, 0), needNormalize(false),
      ranbefore(false) {}
    void addEdge(int s, int e, cost_t cost, cap_t cap)
    {
        if (s == e) return;
        edge forward={e, cost, cap, cap, graph[e].size()};
        edge backward={s, -cost, 0, 0, graph[s].size()};
        if (cost < 0 || ranbefore) needNormalize = true;
        graph[s].emplace_back(forward);
        graph[e].emplace_back(backward);
    }
    bool normalize(int s) {
        auto infinite_cost = numeric_limits<cost_t>::max();
        vector<cost_t> dist(n, infinite_cost);
        dist[s] = 0;
        queue<int> q;
        vector<int> v(n), relax_count(n);
        v[s] = 1; q.push(s);
        while(!q.empty()) {
            int cur = q.front();
            v[cur] = 0; q.pop();
            if (++relax_count[cur] >= n) return false;
            for (const auto &e : graph[cur]) {
                if (iszerocap(e.residual_capacity)) continue;
                auto next = e.target;
                auto ncost = dist[cur] + e.cost;
                if (dist[next] > ncost) {
                    dist[next] = ncost;
                    if (v[next]) continue;
                    v[next] = 1; q.push(next);
                }
            }
        }
        for (int i = 0; i < n; i++) pi[i] = dist[i];
        return true;
    }

    pair<cost_t, cap_t> AugmentShortest(int s, int e, cap_t flow_limit) {
        auto infinite_cost = numeric_limits<cost_t>::max();
        auto infinite_flow = numeric_limits<cap_t>::max();
        typedef pair<cost_t, int> pq_t;
        priority_queue<pq_t, vector<pq_t>, greater<pq_t>> pq;
        vector<pair<cost_t, cap_t>> dist(n, make_pair(infinite_cost, 0));
        vector<int> from(n, -1), v(n);

        if (needNormalize || (ranbefore && lastStart != s))
            normalize(s);
        ranbefore = true;
        lastStart = s;
```

```
        dist[s] = pair<cost_t, cap_t>(0, infinite_flow);
        pq.emplace(dist[s].first, s);
        while(!pq.empty()) {
            auto cur = pq.top().second; pq.pop();
            if (v[cur]) continue;
            v[cur] = 1;
            if (cur == e) continue;
            for (const auto &e : graph[cur]) {
                auto next = e.target;
                if (v[next]) continue;
                if (iszerocap(e.residual_capacity)) continue;
                auto ncost = dist[cur].first + e.cost - pi[next] + pi[cur];
                auto nflow = min(dist[cur].second, e.residual_capacity);
                if (dist[next].first <= ncost) continue;
                dist[next] = make_pair(ncost, nflow);
                from[next] = e.revid;
                pq.emplace(dist[next].first, next);
            }
        }
        /** augment the shortest path **/
        auto p = e;
        auto pathcost = dist[p].first + pi[p] - pi[s];
        auto flow = dist[p].second;
        if (iszerocap(flow)|| (flow_limit <= 0 && pathcost >= 0)) return pair<
          cost_t, cap_t>(0, 0);
        if (flow_limit > 0) flow = min(flow, flow_limit);
        /* update potential */
        for (int i = 0; i < n; i++) {
            if (iszerocap(dist[i].second)) continue;
            pi[i] += dist[i].first;
        }
        while (from[p] != -1) {
            auto nedge = from[p];
            auto np = graph[p][nedge].target;
            auto fedge = graph[p][nedge].revid;
            graph[p][nedge].residual_capacity += flow;
            graph[np][fedge].residual_capacity -= flow;
            p = np;
        }
        return make_pair(pathcost * flow, flow);
    }

    pair<cost_t,cap_t> solve(int s, int e, cap_t flow_minimum = numeric_limits<
      cap_t>::max()) {
        cost_t total_cost = 0;
        cap_t total_flow = 0;
        for(;;) {
            auto res = AugmentShortest(s, e, flow_minimum - total_flow);
            if (res.second <= 0) break;
            total_cost += res.first;
            total_flow += res.second;
        }
        return make_pair(total_cost, total_flow);
    }
};
```

## 5.11 General Min-cut (Stoer-Wagner)

```
// implementation of Stoer-Wagner algorithm
// O(V^3)
//usage
// MinCut mc;
// mc.init(n);
// for (each edge) mc.addEdge(a,b,weight);
// mincut = mc.solve();
// mc.cut = {0,1}^n describing which side the vertex belongs to.
struct MinCutMatrix
{
    typedef int cap_t;
    int n;
    vector<vector<cap_t>> graph;

    void init(int _n) {
        n = _n;
        graph = vector<vector<cap_t>>(n, vector<cap_t>(n, 0));
    }
    void addEdge(int a, int b, cap_t w) {
        if (a == b) return;
        graph[a][b] += w;
        graph[b][a] += w;
    }

    pair<cap_t, pair<int, int>> stMinCut(vector<int> &active) {
        vector<cap_t> key(n);
        vector<int> v(n);
        int s = -1, t = -1;
        for (int i = 0; i < active.size(); i++) {
            cap_t maxv = -1;
            int cur = -1;
            for (auto j : active) {
                if (v[j] == 0 && maxv < key[j]) {
                    maxv = key[j];
                    cur = j;
                }
            }
            t = s; s = cur;
            v[cur] = 1;
            for (auto j : active) key[j] += graph[cur][j];
        }
        return make_pair(key[s], make_pair(s, t));
    }

    vector<int> cut;

    cap_t solve() {
        cap_t res = numeric_limits<cap_t>::max();
        vector<vector<int>> grps;
        vector<int> active;
```

```
        cut.resize(n);
        for (int i = 0; i < n; i++) grps.emplace_back(1, i);
        for (int i = 0; i < n; i++) active.push_back(i);
        while (active.size() >= 2) {
            auto stcut = stMinCut(active);
            if (stcut.first < res) {
                res = stcut.first;
                fill(cut.begin(), cut.end(), 0);
                for (auto v : grps[stcut.second.first]) cut[v] = 1;
            }

            int s = stcut.second.first, t = stcut.second.second;
            if (grps[s].size() < grps[t].size()) swap(s, t);

            active.erase(find(active.begin(), active.end(), t));
            grps[s].insert(grps[s].end(), grps[t].begin(), grps[t].end());
            for (int i = 0; i < n; i++) { graph[i][s] += graph[i][t]; graph[i][t] = 0; }
            for (int i = 0; i < n; i++) { graph[s][i] += graph[t][i]; graph[t][i] = 0; }
            graph[s][s] = 0;
        }
        return res;
    }
};
```

# 6 Geometry

## 6.1 Basic Operations

```
const double eps = 1e-9;

inline int diff(double lhs, double rhs) {
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}

inline bool is_between(double check, double a, double b) {
    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}

struct Point {
    double x, y;
    bool operator==(const Point& rhs) const {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    Point operator+(const Point& rhs) const {
        return Point{ x + rhs.x, y + rhs.y };
    }
    Point operator-(const Point& rhs) const {
        return Point{ x - rhs.x, y - rhs.y };
    }
    Point operator*(double t) const {
        return Point{ x * t, y * t };
    }
};

struct Circle {
    Point center;
    double r;
};

struct Line {
    Point pos, dir;
};

inline double inner(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}

inline double outer(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

inline int ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}

inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}

inline double dist(const Point& a, const Point& b) {
    return sqrt(inner(a - b, a - b));
}

inline double dist2(const Point &a, const Point &b) {
    return inner(a - b, a - b);
}

inline double dist(const Line& line, const Point& point, bool segment = false) {
    double c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    double c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1) <= 0) return dist(line.pos + line.dir, point);
    return dist(line.pos + line.dir * (c1 / c2), point);
}

bool get_cross(const Line& a, const Line& b, Point& ret) {
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    ret = b.pos + b.dir * t2;
    return true;
}
```

```
79
80 bool get_segment_cross(const Line& a, const Line& b, Point& ret) {
81     double mdet = outer(b.dir, a.dir);
82     if (diff(mdet, 0) == 0) return false;
83     double t1 = -outer(b.pos - a.pos, b.dir) / mdet;
84     double t2 = outer(a.dir, b.pos - a.pos) / mdet;
85     if (!is_between(t1, 0, 1) || !is_between(t2, 0, 1)) return false;
86     ret = b.pos + b.dir * t2;
87     return true;
88 }
89
90 Point inner_center(const Point &a, const Point &b, const Point &c) {
91     double wa = dist(b, c), wb = dist(c, a), wc = dist(a, b);
92     double w = wa + wb + wc;
93     return Point{ (wa * a.x + wb * b.x + wc * c.x) / w, (wa * a.y + wb * b.y +
        wc * c.y) / w };
94 }
95
96 Point outer_center(const Point &a, const Point &b, const Point &c) {
97     Point d1 = b - a, d2 = c - a;
98     double area = outer(d1, d2);
99     double dx = d1.x * d1.x * d2.y - d2.x * d2.x * d1.y
100        + d1.y * d2.y * (d1.y - d2.y);
101    double dy = d1.y * d1.y * d2.x - d2.y * d2.y * d1.x
102        + d1.x * d2.x * (d1.x - d2.y);
103    return Point{ a.x + dx / area / 2.0, a.y - dy / area / 2.0 };
104 }
105
106 vector<Point> circle_line(const Circle& circle, const Line& line) {
107     vector<Point> result;
108     double a = 2 * inner(line.dir, line.dir);
109     double b = 2 * (line.dir.x * (line.pos.x - circle.center.x)
110         + line.dir.y * (line.pos.y - circle.center.y));
111     double c = inner(line.pos - circle.center, line.pos - circle.center)
112         - circle.r * circle.r;
113     double det = b * b - 2 * a * c;
114     int pred = diff(det, 0);
115     if (pred == 0)
116         result.push_back(line.pos + line.dir * (-b / a));
117     else if (pred > 0) {
118         det = sqrt(det);
119         result.push_back(line.pos + line.dir * ((-b + det) / a));
120         result.push_back(line.pos + line.dir * ((-b - det) / a));
121     }
122     return result;
123 }
124
125 vector<Point> circle_circle(const Circle& a, const Circle& b) {
126     vector<Point> result;
127     int pred = diff(dist(a.center, b.center), a.r + b.r);
128     if (pred > 0) return result;
129     if (pred == 0) {
130         result.push_back((a.center * b.r + b.center * a.r) * (1 / (a.r + b.r)));
131         return result;
132     }
133     double aa = a.center.x * a.center.x + a.center.y * a.center.y - a.r * a.r;
134     double bb = b.center.x * b.center.x + b.center.y * b.center.y - b.r * b.r;
135     double tmp = (bb - aa) / 2.0;
136     Point cdiff = b.center - a.center;
137     if (diff(cdiff.x, 0) == 0) {
138         if (diff(cdiff.y, 0) == 0)
139             return result; // if (diff(a.r, b.r) == 0): same circle
140         return circle_line(a, Line{ Point{ 0, tmp / cdiff.y }, Point{ 1, 0 } });
141     }
142     return circle_line(a,
143         Line{ Point{ tmp / cdiff.x, 0 }, Point{ -cdiff.y, cdiff.x } });
144 }
145
146 Circle circle_from_3pts(const Point& a, const Point& b, const Point& c) {
147     Point ba = b - a, cb = c - b;
148     Line p{ (a + b) * 0.5, Point{ ba.y, -ba.x } };
149     Line q{ (b + c) * 0.5, Point{ cb.y, -cb.x } };
150     Circle circle;
151     if (!get_cross(p, q, circle.center))
152         circle.r = -1;
153     else
154         circle.r = dist(circle.center, a);
155     return circle;
156 }
157
158 Circle circle_from_2pts_rad(const Point& a, const Point& b, double r) {
159     double det = r * r / dist2(a, b) - 0.25;
160     Circle circle;
161     if (det < 0)
162         circle.r = -1;
163     else {
164         double h = sqrt(det);
165         // center is to the left of a->b
166         circle.center = (a + b) * 0.5 + Point{ a.y - b.y, b.x - a.x } * h;
167         circle.r = r;
168     }
169     return circle;
170 }
```

## 6.2 Compare angles

## 6.3 Convex Hull

```
1 // find convex hull
2 // O(n*logn)
3 vector<Point> convex_hull(vector<Point>& dat) {
4     if (dat.size() <= 3) return dat;
5     vector<Point> upper, lower;
6     sort(dat.begin(), dat.end(), [](const Point& a, const Point& b) {
7         return (a.x == b.x) ? a.y < b.y : a.x < b.x;
8     });
9     for (const auto& p : dat) {
10        while (upper.size() >= 2 && ccw(*++upper.rbegin(), *upper.rbegin(), p)
            >= 0) upper.pop_back();
```

```
        while (lower.size() >= 2 && ccw(*++lower.rbegin(), *lower.rbegin(), p)
          <= 0) lower.pop_back();
        upper.emplace_back(p);
        lower.emplace_back(p);
    }
    upper.insert(upper.end(), ++lower.rbegin(), --lower.rend());
    return upper;
}
```

## 6.4 Polygon Cut

```
// left side of a->b
vector<Point> cut_polygon(const vector<Point>& polygon, Line line) {
    if (!polygon.size()) return polygon;
    typedef vector<Point>::const_iterator piter;
    piter la, lan, fi, fip, i, j;
    la = lan = fi = fip = polygon.end();
    i = polygon.end() - 1;
    bool lastin = diff(ccw_line(line, polygon[polygon.size() - 1]), 0) > 0;
    for (j = polygon.begin() ; j != polygon.end() ; j++) {
        bool thisin = diff(ccw_line(line, *j), 0) > 0;
        if (lastin && !thisin) {
            la = i;
            lan = j;
        }
        if (!lastin && thisin) {
            fi = j;
            fip = i;
        }
        i = j;
        lastin = thisin;
    }
    if (fi == polygon.end()) {
        if (!lastin) return vector<Point>();
        return polygon;
    }
    vector<Point> result;
    for (i = fi ; i != lan ; i++) {
        if (i == polygon.end()) {
            i = polygon.begin();
            if (i == lan) break;
        }
        result.push_back(*i);
    }
    Point lc, fc;
    get_cross(Line{ *la, *lan - *la }, line, lc);
    get_cross(Line{ *fip, *fi - *fip }, line, fc);
    result.push_back(lc);
    if (diff(dist2(lc, fc), 0) != 0) result.push_back(fc);
    return result;
}
```

## 6.5 Pick's theorem

격자점으로 구성된 simple polygon이 주어짐. $i$는 polygon 내부의 격자점 수, $b$는 polygon 선분 위 격자점 수, $A$는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다.

$A = i + \frac{b}{2} - 1$

# 7 String

## 7.1 KMP

```
typedef vector<int> seq_t;

void calculate_pi(vector<int>& pi, const seq_t& str) {
    pi[0] = -1;
    for (int i = 1, j = -1; i < str.size(); i++) {
        while (j >= 0 && str[i] != str[j + 1]) j = pi[j];
        if (str[i] == str[j + 1])
            pi[i] = ++j;
        else
            pi[i] = -1;
    }
}

// returns all positions matched
// O(|text|+|pattern|)
vector<int> kmp(const seq_t& text, const seq_t& pattern) {
    vector<int> pi(pattern.size()), ans;
    if (pattern.size() == 0) return ans;
    calculate_pi(pi, pattern);
    for (int i = 0, j = -1; i < text.size(); i++) {
        while (j >= 0 && text[i] != pattern[j + 1]) j = pi[j];
        if (text[i] == pattern[j + 1]) {
            j++;
            if (j + 1 == pattern.size()) {
                ans.push_back(i - j);
                j = pi[j];
            }
        }
    }
    return ans;
}
```

## 7.2 Aho-Corasick

```
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

```

```cpp
struct AhoCorasick
{
    const int alphabet;
    struct node {
        node() {}
        explicit node(int alphabet) : next(alphabet) {}
        vector<int> next, report;
        int back = 0, output_link = 0;
    };
    int maxid = 0;
    vector<node> dfa;
    explicit AhoCorasick(int alphabet) : alphabet(alphabet), dfa(1, node(alphabet)) { }
    template<typename InIt, typename Fn> void add(int id, InIt first, InIt last, Fn func) {
        int cur = 0;
        for ( ; first != last; ++first) {
            auto s = func(*first);
            if (auto next = dfa[cur].next[s]) cur = next;
            else {
                cur = dfa[cur].next[s] = (int)dfa.size();
                dfa.emplace_back(alphabet);
            }
        }
        dfa[cur].report.push_back(id);
        maxid = max(maxid, id);
    }
    void build() {
        queue<int> q;
        vector<char> visit(dfa.size());
        visit[0] = 1;
        q.push(0);
        while(!q.empty()) {
            auto cur = q.front(); q.pop();
            dfa[cur].output_link = dfa[cur].back;
            if (dfa[dfa[cur].back].report.empty())
                dfa[cur].output_link = dfa[dfa[cur].back].output_link;
            for (int s = 0; s < alphabet; s++) {
                auto &next = dfa[cur].next[s];
                if (next == 0) next = dfa[dfa[cur].back].next[s];
                if (visit[next]) continue;
                if (cur) dfa[next].back = dfa[dfa[cur].back].next[s];
                visit[next] = 1;
                q.push(next);
            }
        }
    }
    template<typename InIt, typename Fn> vector<int> countMatch(InIt first, InIt last, Fn func) {
        int cur = 0;
        vector<int> ret(maxid+1);
        for (; first != last; ++first) {
            cur = dfa[cur].next[func(*first)];
            for (int p = cur; p; p = dfa[p].output_link)
                for (auto id : dfa[p].report) ret[id]++;
        }
        return ret;
    }
};
```

## 7.3 Suffix Array with LCP

```cpp
typedef char T;

// calculates suffix array.
// O(n*logn)
vector<int> suffix_array(const vector<T>& in) {
    int n = (int)in.size(), c = 0;
    vector<int> temp(n), pos2bckt(n), bckt(n), bpos(n), out(n);
    for (int i = 0; i < n; i++) out[i] = i;
    sort(out.begin(), out.end(), [&](int a, int b) { return in[a] < in[b]; });
    for (int i = 0; i < n; i++) {
        bckt[i] = c;
        if (i + 1 == n || in[out[i]] != in[out[i + 1]]) c++;
    }
    for (int h = 1; h < n && c < n; h <<= 1) {
        for (int i = 0; i < n; i++) pos2bckt[out[i]] = bckt[i];
        for (int i = n - 1; i >= 0; i--) bpos[bckt[i]] = i;
        for (int i = 0; i < n; i++)
            if (out[i] >= n - h) temp[bpos[bckt[i]]++] = out[i];
        for (int i = 0; i < n; i++)
            if (out[i] >= h) temp[bpos[pos2bckt[out[i] - h]]++] = out[i] - h;
        c = 0;
        for (int i = 0; i + 1 < n; i++) {
            int a = (bckt[i] != bckt[i + 1]) || (temp[i] >= n - h)
                    || (pos2bckt[temp[i + 1] + h] != pos2bckt[temp[i] + h]);
            bckt[i] = c;
            c += a;
        }
        bckt[n - 1] = c++;
        temp.swap(out);
    }
    return out;
}

// calculates lcp array. it needs suffix array & original sequence.
// O(n)
vector<int> lcp(const vector<T>& in, const vector<int>& sa) {
    int n = (int)in.size();
    if (n == 0) return vector<int>();
    vector<int> rank(n), height(n - 1);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    for (int i = 0, h = 0; i < n; i++) {
        if (rank[i] == 0) continue;
        int j = sa[rank[i] - 1];
        while (i + h < n && j + h < n && in[i + h] == in[j + h]) h++;
        height[rank[i] - 1] = h;
        if (h > 0) h--;
    }
```

```
    return height;
}
```

### 7.4 Suffix Tree

### 7.5 Manacher's Algorithm

```
// find longest palindromic span for each element in str
// O(|str|)
void manacher(const string& str, int plen[]) {
    int r = -1, p = -1;
    for (int i = 0; i < str.length(); ++i) {
        if (i <= r)
            plen[i] = min((2 * p - i >= 0) ? plen[2 * p - i] : 0, r - i);
        else
            plen[i] = 0;
        while (i - plen[i] - 1 >= 0 && i + plen[i] + 1 < str.length()
                && str[i - plen[i] - 1] == str[i + plen[i] + 1]) {
            plen[i] += 1;
        }
        if (i + plen[i] > r) {
            r = i + plen[i];
            p = i;
        }
    }
}
```

## 8 Miscellaneous

### 8.1 Fast I/O

```
namespace fio {
    const int BSIZE = 524288;
    char buffer[BSIZE];
    int p = BSIZE;
    inline char readChar() {
        if(p == BSIZE) {
            fread(buffer, 1, BSIZE, stdin);
            p = 0;
        }
        return buffer[p++];
    }
    int readInt() {
        char c = readChar();
        while ((c < '0' || c > '9') && c != '-') {
            c = readChar();
        }
        int ret = 0; bool neg = c == '-';
        if (neg) c = readChar();
        while (c >= '0' && c <= '9') {
            ret = ret * 10 + c - '0';
            c = readChar();
        }
        return neg ? -ret : ret;
    }
}
```

### 8.2 Magic Numbers

소수 : 10 007 , 10 009 , 10 111 , 31 567 , 70 001 , 1 000 003 , 1 000 033 , 4 000 037 , 1 000 000 007 , 1 000 000 009

### 8.3 Java Examples

```
import java.util.Scanner;

public class example
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int T = in.nextInt();
        while (T --> 0)
        {
            String str = in.next();
            if (str.matches("[A-F]?A+F+C+[A-F]?"))
                System.out.println("Infected!");
            else
                System.out.println("Good");
        }
    }
}
```